Faster Fundamental Graph Algorithms via Learned Predictions

Justin Y. Chen
MIT
justc@mit.edu

Sandeep Silwal
MIT
silwal@mit.edu

Ali Vakilian
TTIC
vakilian@ttic.edu

Fred Zhang
UC Berkeley
z0@berkeley.edu

Abstract

We consider the question of speeding up classic graph algorithms with machine-learned predictions. In this model, algorithms are furnished with extra advice learned from past or similar instances. Given the additional information, we aim to improve upon the traditional worst-case run-time guarantees. Our contributions are the following:

(i) We give a faster algorithm for minimum-weight bipartite matching via learned duals, improving the recent result by Dinitz, Im, Lavastida, Moseley and Vassilvitskii (NeurIPS, 2021);

(ii) We extend the learned dual approach to the single-source shortest path problem (with negative edge lengths), achieving an almost linear runtime given sufficiently accurate predictions which improves upon the classic fastest algorithm due to Goldberg (SIAM J. Comput., 1995);

(iii) We provide a general reduction-based framework for learning-based graph algorithms, leading to new algorithms for degree-constrained subgraph and minimum-cost 0-1 flow, based on reductions to bipartite matching and the shortest path problem.

Finally, we give a set of general learnability theorems, showing that the predictions required by our algorithms can be efficiently learned in a PAC fashion.

1 Introduction

There has been recent interest in moving beyond the traditional and often pessimistic worst-case analysis of algorithms by using machine-learned predictions. This paradigm of *learning-augmented algorithms* is inspired by the great success of machine learning (ML) and aims to utilize ML predictions to improve the performance of classic algorithms.

The extra information assumed in learning-augmented algorithms can be supplied in a variety of settings. For example, in data streams, the observation that underlying patterns in real-world datasets do not change quickly over time has led to the development of an oracle capable of predicting frequently occurring stream elements. In distribution learning, it is natural to have access to different but related distributions that can aid our learning tasks. In many other applications, the current input can be similar to past instances that might help us to avoid computing the solution from scratch. All these scenarios fall under the general umbrella of a "warm start", which enables better initialization of the algorithms to improve their performance.

This learning-based paradigm has been successfully applied in many algorithmic domains. They all share an underlying goal to minimize some resource constraints: in online algorithms, predictions are used to make better future decisions and reduce regret and competitive ratios [LV18]. In streaming algorithms and data structures, predictors have been developed to optimize space usage [KBC+18, HIKV19]. In sublinear algorithms, predictors can reduce the sample complexity of a task [EIN+21].

Despite this activity, only recently have there been works on provably improving the time complexity of algorithms under this framework. Two recent works which consider this resource include the work of [EFS+21] on the k-means clustering problem and the work of [DIL+21] on graph matching; our paper relates to the latter. In [DIL+21], the authors give a learning-based algorithm for (min-cost) bipartite matching and show that predictions provably result in faster algorithms. Our paper is motivated by three natural follow-up questions:

(i) Can we derive learning-augmented algorithms which exploit warm starts and other auxilliary information for other important graph optimization problems besides bipartite matching?

(ii) Do we need a tailor made learning-augmented algorithm for every different graph optimization problem?

(iii) Can we understand when warm starts are learnable for general problems?

1.1 Our Results

Our main contributions provide answers to the three motivating questions. We individually address these questions and our relevant contributions.

> *Can we derive learning-augmented algorithms for other classic graph optimization problems besides bipartite matching?*

Towards answering this question, we first provide a more efficient learning-augmented algorithm for bipartite matching than the one in [DIL+21], in Section 3. The algorithm of [DIL+21] uses dual variables from the linear programming formulation of bipartite matching as predictions. We achieve better runtime by utilizing the interplay of this dual and another set of related dual variables, called reduced edge duals, arising from viewing bipartite matching as a max-flow problem. This result extends to b-matching as well.

Theorem 1.1 (Informal; see Theorem 3.2)**.** *Given a weighted bipartite graph and predicted dual $\hat{y}$, there exists an algorithm that finds a minimum weight perfect matching in time $O(m\sqrt{n}+(m+n\log n)||y^*-\hat{y}||_0)$, where y^* is an optimal dual solution.*

This significantly improves upon the prior bound of $\widetilde{O}(\min\{m\sqrt{n}||y^*-\hat{y}'||_1, mn\})$ due to [DIL+21].

Beyond the problem of minimum-weight matching, we also use reduced edge duals, which allow us to obtain the first learning-augmented algorithm for the single-source shortest-paths problem with negative edge lengths.

Theorem 1.2 (Informal; see Theorem D.4)**.** *Given a directed graph with negative edge weights and predicted dual $\hat{y}$, there exists an algorithm that finds single-source shortest paths in time $O(m \min\{\|\hat{y} - y^*\|_1 \cdot \|\hat{y} - y^*\|_\infty, \sqrt{n} \log(\|\hat{y} - y^*\|_\infty)\})$, where y^* is an optimal dual solution.*

To properly utilize these duals, we give an efficient rounding scheme which takes in as input a set of predicted reduced edge duals and rounds them to a feasible instance. See Section 4 for the formal notion of feasibility and our rounding algorithm.

Do we need a 'tailor made' learning-augmented algorithm for every different graph optimization problem?

The prior work [DIL+21] outlined the three challenges of "feasibility, optimization, and learnabilty" needed to put warm start heuristics on theoretical footing. We leverage *reductions* to avoid addressing these challenges from scratch for each new graph problem. Specifically, we introduce a general framework of reductions that takes existing learning-augmented algorithms and applies them to new problems. Note that in the context of learning-augmented algorithms, we need reductions that efficiently convert instances of a given problem to instances of another problem which we know how to solve using predictions. Therefore, we must judiciously choose the problems and reductions to apply in this framework. Nonetheless, the benefits of our reduction framework include faster learning-augmented algorithms for shortest-paths and new algorithms for other problems, such as degree-constrained subgraph and unit-capacity maximum flow.

Can we understand when warm starts are learnable for general graph problems?

Given the wide range of problems we consider, we need to understand when good hints and predictions which generalize are learnable. (Note that [DIL+21] is only concerned about learnability of duals for the specific problem of bipartite matching.) We answer this question in Section 6 by generalizing the arguments of [DIL+21] beyond bipartite matching.

1.2 Related Work

Learning-augmented graph algorithms. The most relevant work to ours is [DIL+21]. We improve and extend their results in several ways, as discussed earlier. For the shortest-path problem, a recent work [EIX22] investigates the theory of learning-based labeling scheme for the A^* search heuristic, whereas a few others approach it empirically [BCS17, YTB+21, CLDS20]. Several previous papers focus on *online* graph problems [AGKK20, LMRX21, XM21, APT22]. The scope of our paper differs from theirs, as we study only offline problems.

Classic graph algorithms. There is a vast body of literature addressing graph optimization problems considered in this paper. We only mention a few that are most relevant to this paper. Similar to [DIL+21], our matching algorithm builds upon the classic Hungarian method. There are other theoretically faster (exact) algorithms for (bipartite) minimum-cost perfect matching, including [OA92, GK97, DS12]. However, these procedures are fairly involved and hard to incorporate predictions. Our learning-based algorithm for single-source negative-length shortest paths is inspired by [Gol95]. In the classic setting, a web of reductions among graph problems were introduced by [Gab83, Gab85, GT89].

The learning-based algorithm paradigm has been applied to a number of other problems. See Appendix A for more related works.

1.3 Organization

The remainder of the paper proceeds as follows. We set up some preliminary background and notations in Section 2. In Section 3, we give an improved algorithm for learning-augmented minimum-cost bipartite matching. We then extend the approach to shortest path in Section 4. We address the question of learnability in Section 6. Finally, we provide numerical evaluations of our shortest-path algorithm via reductions in Section 7.

2 Preliminaries

Notation. Let $G = (V, E)$ be a graph of m edges and n vertices. We will specify its directedness in different settings. For a vector $x \in \mathbb{R}^m$, we let $\|x\|_p$ to denote its pth norm, for any $p \geq 0$.

Minimum-Weight Bipartite Perfect Matching. Let $G = (V, E)$ be a bipartite graph with non-negative edge costs, and C be the maximum cost. The objective of this problem is to find a perfect matching M with minimum total cost in G. In the minimum-weight b-matching problem, we are also given a demand vector $b \in \mathbb{Z}_+^V$. The goal is to match each vertex u b_u times, with minimum cost.

Maximum Flow. Given a directed graph $G = (V, E)$ with capacity vector $c \in \mathbb{R}_+^E$, let s and t be distinct vertices of H. A feasible s-t flow is a vector $f \in \mathbb{R}_+^E$, with each entry representing flow along an edge, such that sum of incoming flow along edges (v, u) equals sum of outgoing flow along edges (u, v) for all $u \in V \setminus \{s, t\}$ and $f_{uv} \leq c_{uv}$ for all edge uv. An s-t-flow f is maximum if it maximizes the outgoing flow from s. For a feasible flow f, G_f denotes its residual graph. A classic procedure for finding maximum flow is Ford-Fulkerson; see [CLRS09].

3 Improved Learning-Based Minimum-Weight Matching

The results from [DIL+21] on matching contain two main results: (1) that given predicted duals for a minimum-weight matching problem, there is an efficient near-optimal algorithm to round the duals to feasibility and (2) that after rounding, these feasible predicted duals can be used to quickly find a solution to the primal. We provide a new approach to the second problem of using a feasible prediction to quickly solve minimum-weight matching that significantly improves upon prior work.

First, we will restate the theorem from [DIL+21] which established an algorithm from using a feasible predicted dual to quickly solve minimum-weight matching.

Theorem 3.1 (Theorem 13 in [DIL+21])**.** *There exists an algorithm which takes as input a feasible integer dual solution $\hat{y}'$ and finds a minimum-weight bipartite perfect matching in $\widetilde{O}(\min\{m\sqrt{n}\|y^* - \hat{y}'\|_1, mn\})$ time, where y^* is an optimal dual solution.*

With small modifications to the algorithm and an improved analysis, we present the following improved time complexity.

Theorem 3.2 (Faster Matching from Predicted Duals)**.** *There exists an algorithm which takes as input a feasible integer dual solution $\hat{y}'$ and finds a minimum-weight bipartite perfect matching in $O(m\sqrt{n} + (m + n \log n)\|y^* - \hat{y}'\|_0)$ time, where y^* is an optimal dual solution.*

If the feasible dual is within $O(\sqrt{n})$ of an optimal dual in ℓ_1 distance (which is the case in which the algorithm from [DIL+21] attains an improved runtime over the classical algorithm), our algorithm improves upon the time complexity by a factor of

$$\sqrt{n}\left(\frac{\|y^* - \hat{y}'\|_1}{\|y^* - \hat{y}'\|_0}\right).$$

Note that $\|y^*(c) - \hat{y}'(c)\|_0 < \|y^*(c) - \hat{y}'(c)\|_1$ as we are considering only integral duals.

While the algorithm from [DIL+21] improves upon the classic Hungarian algorithm only when $\|y^*(c) - \hat{y}'(c)\|_1 = o(\sqrt{n})$, our algorithm improves upon the Hungarian algorithm as long as $\|y^*(c) - \hat{y}'(c)\|_0 = o(n)$, a much milder condition on the predictions.

As a corollary, when combined with the linear-time rounding procedure from [DIL+21], this algorithm gives a fast framework for taking a predicted (possibly infeasible) dual and using it to speed up minimum-weight matching.

Corollary 3.3. *There exists an algorithm which takes as input a (possibly infeasible) integral dual solution $\hat{y}$, produces a feasible dual $\hat{y}'$ s.t. $\|\hat{y}' - y^*\|_1 \leq 3\|\hat{y} - y^*\|_1$, and finds a minimum-weight bipartite perfect matching in $O(m\sqrt{n} + (m + n\log n)\|y^* - \hat{y}'\|_0)$ time, where y^* is an optimal dual solution.*

Our algorithm is given in Algorithm 1. The main difference in the algorithm/analysis to prior work is that they essentially consider running a $O(m\sqrt{n})$ matching algorithm at each step and then reason that the dual variables increase by at least one on each call to the algorithm, getting the ℓ_1 dependence on the error.

Our improvements are based on the following observation. If the predicted duals are accurate enough to get improvements over the normal Hungarian algorithm, then the first call to a maximum cardinality matching algorithm will match many edges. Then, we can account for the amount of work we have to do in subsequent iterations by the small number of edges remaining to be matched by via a flow interpretation of the matching problem.

Algorithm 1 Faster Primal-Dual Scheme for MWPM

procedure MWPM-PD++($G = (L \cup R, E), c, y$)
  $E' \leftarrow \{e \in E \mid y_i + y_j = c_{ij}\ \}$
  $G' \leftarrow (V, E')$
  $M \leftarrow$ Maximum cardinality matching in G'
  Give all edges in E unit capacity and direct them from left to right ▷ Flow representation
  Add nodes s, t to G along with unit capacity, zero cost edges (s, i) for all $i \in L$ and (j, t) for all $j \in R$
  Associate a flow f with M s.t. $\forall (i, j) \in M$, $f_{si} = f_{ij} = f_{jt} = 1$ and otherwise $f_e = 0$
  $z_i \leftarrow -y_i \quad \forall i \in L$
  $z_j \leftarrow y_j \quad \forall j \in R$
  $c'_e \leftarrow c_e + z_i - z_j \quad \forall e = (i, j) \in E$ s.t. $i, j \notin \{s, t\}$ and $c'_e \leftarrow 0$ for all other edges
  while f has flow value less than n **do**
    $z_u \leftarrow z_u + d(s, u)\ \forall u \in L \cup R$ where $d(\cdot, \cdot)$ is shortest path distance in G_f w.r.t. c' ▷ Dijkstra
    $c'_e \leftarrow c_e + z_i - z_j \quad \forall e = (i, j) \in E$ s.t. $i, j \notin \{s, t\}$
    $E'_f \leftarrow \{e \in E_f | c'_e = 0\}$.
    $G'_f \leftarrow (V, E'_f)$
    $g \leftarrow$ Maximum flow in G'_f ▷ Ford-Fulkerson
    Augment along g in G_f
  end while
  Return $\{e = (i, j) \in f : i \in L, j \in R, f_e = 1\}$
end procedure

The formal analysis of the algorithm is somewhat technical and appears in Appendix B, where we prove Theorem 3.2.

Extension to b-matching We extend the improvements for learning-based minimum-weight perfect bipartite matching to the more general problem of minimum-weight perfect b-matching. For two sets of dual variables over the vertices y and z, we will use as a distance measure the weighted ℓ_p error:

$$\|y - z\|_{b,p} = \sum_{i \in V} b_i |y_i - z_i|_p.$$

We will restate a theorem from [DIL+21].

Theorem 3.4 (Theorem 31 in [DIL+21])**.** *There exists an algorithm which takes as input a feasible integer dual solution $\hat{y}'$ and finds a minimum-weight perfect b-matching in $O(mn\|y^* - \hat{y}'\|_{b,1})$ time, where y^* is an optimal dual solution.*

Using the same algorithm (Algorithm 4 shown in Appendix C), but with an improved analysis, we show the following improved runtime.

Theorem 3.5. *There exists an algorithm which takes as input a feasible integer dual solution $\hat{y}'$ and finds a minimum-weight perfect b-matching in $O(mn + m\|y^* - \hat{y}'\|_{b,0})$ time, where y^* is an optimal dual solution.*

As before, since the duals are integral, $\|y^* - \hat{y}'\|_{b,0} \leq \|y^* - \hat{y}'\|_{b,1}$. Note that this runtime improves upon prior work by a factor of

$$\min\left\{n\frac{\|y^* - \hat{y}'\|_{b,1}}{\|y^* - \hat{y}'\|_{b,0}}, \|y^* - \hat{y}'\|_{b,1}\right\}.$$

The full details and proof are in Appendix C.

4 Fast Learning-Based Shortest Paths

In this section, we introduce the reduced edge length duals and how to round them efficiently given predictions. Reduced edge length duals are defined as follows.

Definition 4.1 (Reduced Edge Length Duals (RE Duals))**.** *Let $G = (V, E)$ with $|V| = n, |E| = m$, denote a directed graph and $\ell : E \to \mathbb{Z}$ denote the length of the edges, which may be negative. $y \in \mathbb{Z}^V$ is a valid or feasible reduced edge length dual (RE Dual) if*

$$\ell_y(u, v) := \ell(u, v) + y_u - y_v \geq 0$$

for all edges $e = (u, v) \in E$.

It is natural to study these duals as they appear in many fundamental combinatorial optimization problems. For example, consider the dual linear program for the shortest paths problem on the graph G (where we wish to compute the shortest path from vertex s to t). It is given by:

$$\begin{aligned} \max \quad & y_t \\ \text{s.t.} \quad & y_v - y_u \leq \ell(u, v) \\ & y_s = 0. \end{aligned}$$

Note that the constraints $y_v - y_u \leq \ell(u, v)$ exactly encode $\ell_y(u, v) \geq 0$ in Definition 4.1. Furthermore, given a valid dual solution y to the dual linear program, one can quickly compute the shortest paths in near linear time via an application of Dijkstra's algorithm since all reduced edge lengths are non negative by Definition 4.1. This is because the sum of the lengths of edges along any path $(v_1, v_2, \cdots, v_k)$ is the same up to an additive term $y_{v_1} - y_{v_k}$ due to telescoping. Thus, this transformation preserves the identity of shortest paths from a starting vertex. Furthermore, many shortest paths algorithms on general graphs, such as the Bellman-Ford algorithm, also implicitly calculate the dual y: in the Bellman-Ford algorithm, the dual can be constructed in linear time after the algorithm terminates.

Now suppose predictions $\hat{y} : V \to \mathbb{Z}$ for the duals y are given. The main result of this section is that there exists an efficient algorithm, Algorithm 2, which outputs a feasible $\hat{y}'$ according to Definition 4.1.

Theorem 4.1 (Fast Shortest-Path from Predicted Duals)**.** *Let $\hat{y} : V \to \mathbb{Z}$ be predicted duals and let $y^* : V \to \mathbb{Z}$ be a feasible set of reduced edge length duals according to Definition 4.1 such that $\|\hat{y} - y^*\|_1$ is minimized. Algorithm 2 returns a feasible $\hat{y}' : V \to \mathbb{Z}$ in time*

$$O(m \min\{\|\hat{y} - y^*\|_1 \cdot \|\hat{y} - y^*\|_\infty, \sqrt{n}\log(\|\hat{y} - y^*\|_\infty)\}).$$

If we define reduced edge lengths according to the predicted dual $\hat{y}$, it is likely that the non-negativity constraint of some edges become violated, i.e., $\ell_{\hat{y}}(e) < 0$. The goal of Algorithm 2 is to modify some coordinates of $\hat{y}$ to fix these negative edge weights. The algorithm uses a key subroutine of Goldberg's algorithm on shortest paths [Gol95]. It proceeds by mending negative edges by reducing the dual value of one of their endpoints. At every iteration, we greedily maximize the number of dual variable which

are updated. The vertices which are updated are picked through a layering structure utilized in [Gol95]. Algorithm 2 presents the formal details.

Note that we implicitly assume the given graph G with edge lengths given by ℓ does not have a negative weight cycle. This is a necessary assumption since otherwise, there exists no valid RE Duals for G: the length of a cycle under any valid dual y must be non-negative by definition but the cost of any cycle is the same under ℓ and ℓ_y due to telescoping which leads to a contradiction if ℓ induces a negative weight cycle.

Algorithm 2 Rounding Predictions for Reduced Edge Length Duals

Input: Graph $G = (V, E)$, predicted duals $\hat{y} : V \to \mathbb{Z}$
procedure ROUND-RE-DUALS($G, \hat{y}$)
  while there exists an edge e such that $\ell_{\hat{y}}(e) < 0$ **do**
    $G^- = (V, E^-) \leftarrow$ subgraph of G that have weight at most 0 under $\ell_{\hat{y}}$
    Contract all strongly connected components in G^- ▷ All edges connecting vertices in the same strongly connected component are 0 [Gol95]
    Add a vertex x to G^- and connect it with zero length edges to all of V
    $L_i \leftarrow \{v \in V | d(x, v) = -i\}$ ▷ d is graph distance in G^- using reduced edge lengths given by $\ell_{\hat{y}}$
    $i^* \leftarrow \arg\max_i |L_i|$
    Lower the value of $\hat{y}_v$ for all vertices in $\cup_{t \geq i^*} L_t$ by 1
  end while
  Return $\hat{y}$
end procedure

The analysis of the algorithm and the proof of Theorem 4.1 appear in Appendix D. The theorem also implies an algorithm for all-pair shortest paths; see Appendix D.2 for details.

5 A General Framework for Learning-Based Reductions

In this section, we introduce a general framework for obtaining learning-augmented algorithms via reductions. Suppose we have an oracle which provides hints or a warm start to instances of problem P_1. If we are instead interested in solving instances of another problem P_2, we can hope to transform our instance at hand to an instance of P_1 in order to utilize the available predictions. If there exists an efficient reduction from P_2 to P_1, we can indeed use this reduction to transform our instance of P_2 to that of P_1, use the hints available for P_1 to efficiently solve our new problem, and use the solution found to solve our original instance of P_2. This will be the basis of our framework for learning-based reductions.

Why is such a framework useful? First, hints might be easier to learn for problem P_1 or there may not be a natural notion of hints for instances of P_2. In addition, there might already exist a learning-based algorithm for P_1 which efficiently utilizes hints. Therefore, using reductions from other problems to P_1 would eliminate the need to create new algorithms and thereby increasing the power and usability of the existing learning-based algorithms.

We formally define reductions as follows.

Definition 5.1 (Reductions)**.** *Let P_1 and P_2 be two problem instances. We say that $R : P_2 \to P_1$ is a reduction from P_2 to P_1 if for any instance $I \in P_2$, $R(I)$ maps to an instance I' of P_2. Furthermore, Furthermore, there exists mapping which takes a solution of I' and converts it to a solution for I.*

Note that the definition of reduction by itself is not quite useful: by the Cook-Levin theorem, any problem in the complexity class P can be reduced to 3SAT. However in this paper, we are interested in *efficient* reductions which take linear or almost linear time in the size of the input. Therefore, such reductions would be extremely fast to execute in practice and the final algorithm of solving instances of I of P_2 via solving instances of P_1 using a learned oracle would overall be faster than solving I with no hints.

5.1 General Framework

Our framework is given in Algorithm 3. Note that there, $\mathcal{A}$ is an existing algorithm which solves instances of problem P_1 using hints given by a predictor $y : P_1 \to \mathbb{R}^d$, i.e., the hints are d dimensional vectors.

Algorithm 3 General Reduction Framework for Learning-Based Algorithms

1: **Input:** Problem instance $I \in P_2$. Predictor $y : P_1 \to \mathbb{R}^d$, reduction $R : P_2 \to P_1$, Algorithm $\mathcal{A}(P_1, y(P_1))$
2: **procedure** REDUCTION-SOLVE($P_1, P_2, y, R, \mathcal{A}$)
3: $I' \leftarrow R(I)$ ▷ Use reduction R to get an instance of P_1
4: $\hat{y} \leftarrow y(I')$ ▷ Get hints for instance I' using predictor y
5: Execute $\mathcal{A}(I', \hat{y})$ ▷ Solve instance I' using hints and learning-based algorithm $\mathcal{A}$.
6: Return solution to I using solution for I' given by $\mathcal{A}(I', \hat{y})$.
7: **end procedure**

Note that Step 6 in Algorithm 3 would depend on the instances P_1, P_2 and the reduction R. In some cases, some post-processing the solution $\mathcal{A}(I', \hat{y})$ could be required. In the examples we study in this paper, both this step and the reduction R are efficient. We give concrete instantiations of Algorithm 3 in Section 5.2.

Note that we still need to understand the learnability of the hints $\hat{y}$ in Step 4 of Algorithm 3: even if there exists an efficient algorithm $\mathcal{A}$ for problem P_1, we might not have a predictor y at hand. Note that in [DIL+21], the question of learnability of predictors was tackled by assuming access to multiple instances of a particular problem class drawn from some distribution. In our case, we might have lots of training data on instances of problem P_2 but our goal is to train a predictor y for P_1 in hopes of utilizing $\mathcal{A}$. To do so, we can just go through the reduction R to get samples of problem instances drawn from P_1. Note that the distribution on these problems will be different than that on P_2. We introduce general learnability results which imply one can learn a good predictor y. For details, see Section 6.

5.2 Reductions and Their Implications

We now present one application of the reduction framework outlined in the previous section. We demonstrate three more reductions, for degree constrained subgraph, minimum-cost 0-1 flow, and graph diameter, in Appendix E.

Shortest Path from Matching. We leverage the following reduction from shortest path (with negative edge lengths) to maximum-weight perfect matching on bipartite graphs, due to [Gab85]. Given a directed graph G with edge lengths given by ℓ, construct a weighted bipartite graph $H = (L, R, E)$.

- For each vertex $u \in G$, make two copies $u_1 \in L$ and $u_2 \in R$.
- For each arc $(u, v) \in G$, create an edge $e = (u_1, v_2)$ of weight $-\ell(e)$ in H.
- Finally, create an edge (u_1, u_2) of weight 0 for each vertex $u \in G$.

Now suppose we find the maximum weight perfect matching of H and its corresponding dual variables y_{u_i} for all $u \in G$ and $i \in \{1, 2\}$. By the construction, we immediately have:

Lemma 5.1. *The maximum weight perfect matching of H has positive weight if and only if the graph G has negative cycles.*

Otherwise, we can let $\pi_u = y_{u_1}$ for each $u \in G$. Then by feasibility of y, we have

$$-\ell(e) \leq y_{u_1} + y_{v_2} \leq \pi_u - \pi_v,$$

for any $e = (u, v) \in G$. It follows that π is a feasible dual for the shortest path problem on G, i.e., it satisfies Definition 4.1.

Observe that the graph H contains $m+n$ edges. By using the run-times for faster matching from [DIL+21] as well as our runtime of Section Section 3, we have the following corollary due to Algorithm 3.

Theorem 5.2. *Given a shortest path problem on input graph $G = (V, E)$ with n vertices and m edges, there exists an algorithm which takes takes as input a predicted dual solution $\hat{y}$ to an instance of maximum weight perfect matching derived from G, near-optimally rounds the dual to a feasible solution $\hat{y}'$, and finds feasible reduced edge length duals for G in time $O(m\sqrt{n} + (m + n \log n)||y^* - \hat{y}'||_0)$.*

Remark 5.1. *Recall that in Section 4 we derived an alternative runtime of*

$$O(m \min\{\|\hat{y} - y^*\|_1 \cdot \|\hat{y} - y^*\|_\infty, \sqrt{n} \log(\|\hat{y} - y^*\|_\infty))\}$$

for the problem of rounding a predicted dual $\hat{y}$ to a feasible dual $\hat{y}'$ to satisfy the reduced edge property of Definition 4.1. These results are incomparable since Theorem 5.2 uses dual predictions for matchings on a transformed graph whereas Theorem 4.1 uses predictions for shortest path duals (RE duals) on the original graph.

6 General Learnability of Hints

We now present two general learnability theorems on vector-valued hints for graph optimization problems. We consider the model where the edge weights (or capacities) are drawn from a distribution, while the vertex set is the same. The goal is to learn a hint vector in $\mathbb{R}^d$ that is close to the *optimal hint* on average (in ℓ_1 or ℓ_∞ norm), given i.i.d. samples of the edge weights. We require no assumption either on the edge weights distribution or on the notion of optimality of a hint. Indeed, the latter needs to depend on particular problems. For a variety of graph problems, though, the optimal hint could be taken as the optimal dual solution of certain LP relaxation.

Throughout the section, we assume that the edge weights c are drawn from an unknown distribution of $\mathcal{D}$. We search for a hint within a range $\mathcal{H} \subseteq \mathbb{R}^d$.

6.1 Learnability from Bounded Pseudo-Dimension

For a fixed problem, given edge weight $c \in \mathbb{R}^m$, and graph G, let $h^*(c)$ denote an optimal hint with respect to the instance (G, c). We consider ℓ_1 and ℓ_∞ loss:

$$\ell_1(h, c) = \|h^*(c) - h\|_1 = \sum_{i=1}^{d} |h_i^*(c) - h_i|\,, \tag{1}$$

$$\ell_\infty(h, c) = \|h^*(c) - h\|_\infty = \max_i |h_i^*(c) - h_i|\,. \tag{2}$$

The goal of the algorithm is to find a hint $\widehat{h} \in \mathcal{H}$ such that the expected loss $\mathbb{E}_{c\sim\mathcal{D}}\, \ell(h, c)$ is minimized, for $\ell = \ell_1$ or ℓ_∞. Let $h^* \in \arg\min_{h\in\mathcal{H}} \mathbb{E}_{c\sim\mathcal{D}}\, \ell(h, c)$.

***L_1* Loss.** Our first result is a straightforward abstraction of the main learnability theorem of [DIL+21], for the ℓ_1 loss. In particular, we show that one can find a hint vector $\widehat{h} \in \mathbb{R}^d$ that approximately minimizes the population loss $\mathbb{E}_{c\sim\mathcal{D}}\, \ell_1(h, c)$, under the following conditions:

Theorem 6.1 (ℓ_1-learnability; see also Theorem 14 of [DIL+21])**.** *For any graph problem with optimal hint $h^*(c) \in \mathcal{H}$ for $c \sim \mathcal{D}$, assume that*

- *(bounded range) for any $h \in \mathcal{H}$ we have $h_i \in [-M, M]$ for all i, for some M; and*
- *(efficient optimization) there exists a polynomial time algorithm that finds a hint vector $h \in \mathcal{H}$ that minimizes $\sum_{i=1}^{s} \|h^*(c_i) - h\|_1$, given i.i.d. samples $c_1, c_2, ..., c_s \sim \mathcal{D}$.*

Then there is a polynomial-time algorithm that given $s = O\left(\left(\frac{dM}{\epsilon}\right)^2 (d \log d + \log(1/\delta))\right)$ samples returns a hint $h \in \mathcal{H}$ such that $\mathbb{E}_{c\sim\mathcal{D}}\, \ell_1(h, c) \le \mathbb{E}_{c\sim\mathcal{D}}\, \ell_1(h^*(c), c) + \epsilon$ with probability at least $1 - \delta$.

L_∞ Loss. We now give a learnability result for the ℓ_∞ loss.

Theorem 6.2 (ℓ_∞-learnability)**.** *For any graph problem with optimal hint $h^*(c) \in \mathcal{H}$ for $c \sim \mathcal{D}$, assume that*

- *(bounded range) for any $h \in \mathcal{H}$ we have $h_i \in [-M, M]$ for all i, for some M; and*
- *(efficient optimization) there exists a polynomial time algorithm that finds a hint vector $h \in \mathcal{H}$ that minimizes $\sum_{i=1}^{s} \|h^*(c_i) - h\|_\infty$, given i.i.d. samples $c_1, c_2, ..., c_s \sim \mathcal{D}$.*

Then there is a polynomial-time algorithm that given $s = O\left(\left(\frac{dM}{\epsilon}\right)^2 (d + \log(1/\delta))\right)$ samples returns a hint $h \in \mathcal{H}$ such that $\mathbb{E}_{c\sim\mathcal{D}}\, \ell_\infty(h, c) \leq \mathbb{E}_{c\sim\mathcal{D}}\, \ell_\infty(h^*(c), c) + \epsilon$ with probability at least $1 - \delta$.

The proofs of the two theorems appear in Appendix F.

6.2 Learnability from Arithmetic Complexity

We give an alternative argument for learning predictions. Informally, we show that good predictions can be learned efficiently if the loss function can be 'computed efficiently'. This provides a more general framework that goes beyond ℓ_1 or ℓ_∞ norm error. See Appendix F.3 for formal details.

7 Empirical Simulations

We demonstrate the applicability of our learning-based reductions framework with a real world case study on foreign exchange markets. The reduction we focus on is the general shortest paths to bipartite matching reduction outlined in Section 7. We focus our evaluation on this task since prior work in [DIL+21] has already demonstrated the empirical advantage of learning-based methods for bipartite matching.

Our graph dataset is constructed as follows. We have a weighted directed graph where nodes represent countries and all possible directed edges between all pairs are present. The weight of the directed edge from country i and j represents the average monthly exchange rate between the currencies of the two countries, i.e., the amount of currency j we can obtain starting from one unit of currency i, as set by the foreign exchange rate market[1]. We transform these weights by taking the natural logarithm and negating the weight. This implies that the shortest path from country i to country j on the transformed graph represents the *optimal* way to convert one unit of currency i to the currency of j, i.e., the set of conversions which maximize the amount of currency j.

Experimental Setup. We first describe our training dataset. We construct the graph described above for each month of the year 2019 where we use the average monthly exchange rates as edge weights before performing the transformation. Our testing dataset are similarly constructed graphs for each month of 2020 and 2021. For each graph, we construct the reduction from shortest paths to matching outlined in Section 5.2. By Lemma 5.1, the output of the maximum weight perfect matching on the bipartite graph obtained via the reduction gives us feasible reduced edge length duals which we can be subsequently use to solve shortest paths in nearly linear time. The resulting bipartite graphs have ~ 500 vertices each and $\sim 5 \cdot 10^4$ edges.

We use the code from [DIL+21] [2] for the maximum weight bipartite matching algorithm. As in [DIL+21], we measure the runtime in terms of the number of steps used by the Hungarian algorithm to solve the matching instances derived from our training and test graph datasets when we initialize the algorithm with predicted duals versus when we start the algorithm "from scratch."

We instantiate predictions in two distinct ways, similar to the methodology of [DIL+21]: (a) first, we consider the *batch* version where we compute the optimal dual variables in the training set, take their median, and use these as the predicted dual variables for each of the graphs in the test set. (b) The second method is the *online* version where we use the optimal dual variables from the graph for the prior month in the test set as initialization for the current month.

[1] Dataset scraped from https://fxtop.com/en/historical-exchange-rates.php

[2] Available at https://github.com/tlavastida/LearnedDuals

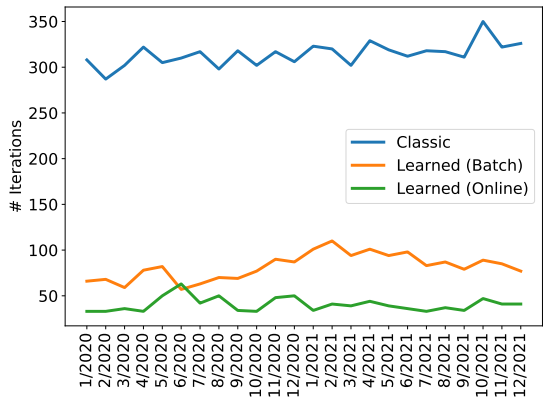

Figure 1: Comparison of the classical Hungarian algorithm (blue) versus learning-augmented algorithms. Predictions lead to up to an order magnitude reduction in number of iterations.

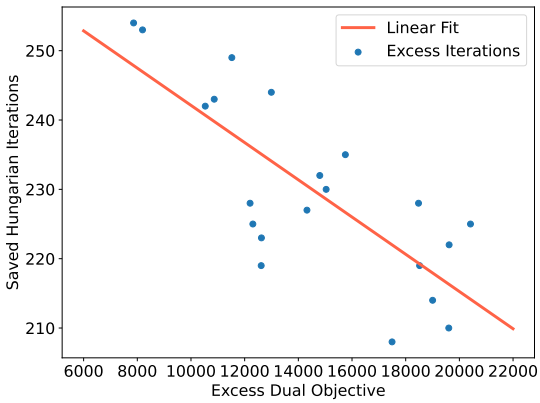

Figure 2: Excess dual objective versus the number of saved Hungarian iterations in the batch version. There is a negative correlation between the excess dual and the # of saved iterations

Results. Our results are shown in Figures 1 and 2. Figure 1 shows up to an *order of magnitude* reduction in the number of iterations taken by the learning-augmented algorithm versus the classical and widely used Hungarian algorithm. As expected, the online method performs slightly better than the batch version as it is able to offer more accurate predictions for the next graph instance. This is very intuitive: it is rare for the foreign exchange market to experience drastic shifts over the span of one month since such a shift implies a major global event.

Our results also validate the dependence of prediction error derived in our theoretical bounds. In Figure 2, we plot the excess dual objective, defined as $\sum_e y_e^* - \sum_e \hat{y}_e$ where recall that y^* represents the optimal dual variables and $\hat{y}$ denotes the predictions, versus the number of steps saved in the Hungarian algorithm in our batch setting; we obtained a qualitatively similar result for the online setting. We see there is a direct linear relationship between the excess dual objective, which represents the prediction error, and the decrease in runtime, measured by the number of Hungarian iterations saved. Note that we removed three outlier points from Figure 2 which represent data from October 2021 to December 2021. The outlier points showed a large excess dual as well as large savings in runtime (which can be inferred from Figure 1). We hypothesize that this is because of a distribution shift which occurred during these months in the foreign exchange markets. Indeed, examining the conversion rate from the Euro to US Dollars for example, we see a 3% decrease in the exchange rate which represents the biggest decrease in the time frame of our training and test dataset. This can be explained by global events such as the rise of the Omicron strain or concerns about increased inflation.

In addition to extending and complementing the experimental results of [DIL+21], we summarize our results in the following points: (a) Our theory is predictive of experimental performance. Both figures demonstrates that better predictions imply better empirical runtime. In addition, Figure 2 demonstrates a direct relationship between prediction error and runtime, as implied by our theoretical bounds. (b) The reduction framework is efficient to carry out and execute in practice. (c) Learning augmented graph algorithms can be applied to real world datasets varying over time such as in the analysis of graphs derived from the foreign exchange rates market.

Acknowledgment

We thank Piotr Indyk for helpful comments on an early draft of the paper and Robert Tarjan for providing us with the reference [Gab83].

Justin and Sandeep are supported by an NSF Graduate Research Fellowship under Grant No. 1745302, NSF TRIPODS program (award DMS-2022448), NSF award CCF-2006798, and Simons Investigator Award. Justin is also supported by a MathWorks Engineering Fellowship. Ali is supported by NSF award CCF-1934843.

References

[ABB99] Martin Anthony, Peter L Bartlett, and Peter L Bartlett. *Neural Network Learning: Theoretical Foundations*, volume 9. Cambridge University Press, 1999.

[ACI22] Anders Aamand, Justin Y. Chen, and Piotr Indyk. (Optimal) Online Bipartite Matching with Predicted Degrees. *CoRR*, abs/2110.11439, 2022.

[AEMP22] Sara Ahmadian, Hossein Esfandiari, Vahab Mirrokni, and Binghui Peng. Robust load balancing with machine learned advice. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2022.

[AGKK20] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *Advanced in Neural Information Processing Systems (NeurIPS)*, 2020.

[AGKP21] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. A regression approach to learning-augmented online algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[AGP20] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ml predictions for online algorithms. In *International Conference on Machine Learning (ICML)*, 2020.

[AKL+19] Daniel Alabi, Adam Tauman Kalai, Katrina Liggett, Cameron Musco, Christos Tzamos, and Ellen Vitercik. Learning to prune: Speeding up repeated computations. In *Conference on Learning Theory (COLT)*, 2019.

[AMK21] Mohammad Ali Alomrani, Reza Moravej, and Elias B Khalil. Deep policies for online bipartite matching: A reinforcement learning approach. *arXiv preprint arXiv:2109.10380*, 2021.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[AMW19] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *International Conference on Learning Representations (ICLR)*, 2019.

[APT22] Yossi Azar, Debmalya Panigrahi, and Noam Touitou. Online graph algorithms with predictions. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.

[BCK+22] Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-augmented weighted paging. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.

[BCS17] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. In *Conference on Robot Learning (CoRL*, 2017.

[BDD+21] Maria-Florina Balcan, Dan F. DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2021.

[BDSV18a] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning (ICML)*, 2018.

[BDSV18b] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning (ICML)*, 2018.

[BDV18] Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

[BDW18] Maria-Florina Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized Lloyd's families. In *Advanced in Neural Information Processing Systems (NeurIPS)*, 2018.

[BLP21] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[BMS20] Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[BPL+16] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

[CGP20] Edith Cohen, Ofir Geri, and Rasmus Pagh. Composable sketches for functions of frequencies: Beyond the worst case. In *International Conference on Machine Learning (ICML)*, 2020.

[CGT+20] Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. Pandora's box with correlations: Learning and approximation. In *IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020.

[CLDS20] Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. Retro*: learning retrosynthetic planning with neural guided a* search. In *International Conference on Machine Learning (ICML)*, 2020.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[DIL+21] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[DIRW20] Yihe Dong, Piotr Indyk, Ilya P Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. *International Conference on Learning Representations (ICLR)*, 2020.

[DKT+21] Ilias Diakonikolas, Vasilis Kontonis, Christos Tzamos, Ali Vakilian, and Nikos Zarifis. Learning online algorithms with distributional advice. In *International Conference on Machine Learning (ICML)*, 2021.

[DS12] Ran Duan and Hsin-Hao Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, 2012.

[DW21] Mina Dalirrooyfard and Nicole Wein. Tight conditional lower bounds for approximating diameter in directed graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2021.

[DWM21] Elbert Du, Franklyn Wang, and Michael Mitzenmacher. Putting the "learning" into learning-augmented algorithms for frequency estimation. In *International Conference on Machine Learning (ICML)*, 2021.

[EFS+21] Jon Ergun, Zhili Feng, Sandeep Silwal, David P. Woodruff, and Samson Zhou. Learning-augmented k-means clustering. *CoRR*, abs/2110.14094, 2021.

[EIN+21] Talya Eden, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, Sandeep Silwal, and Tal Wagner. Learning-based support estimation in sublinear time. In *International Conference on Learning Representations (ICLR)*, 2021.

[EIX22] Talya Eden, Piotr Indyk, and Haike Xu. Embeddings and labeling schemes for a. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, 2022.

[EKM15] Hossein Esfandiari, Nitish Korula, and Vahab Mirrokni. Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation (EC)*, 2015.

[FLV20] Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In *International Conference on Machine Learning (ICML)*, 2020.

[Gab83] Harold N Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC)*, 1983.

[Gab85] Harold N Gabow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148–168, 1985.

[GK97] Andrew V Goldberg and Robert Kennedy. Global price updates help. *SIAM Journal on Discrete Mathematics*, 10(4):551–572, 1997.

[Gol95] Andrew V Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[GP19] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning (ICML)*, 2019.

[GR17] Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM J. Comput.*, 46:992–1017, 2017.

[GT89] Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[HIKV19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations (ICLR)*, 2019.

[IKMQP21] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2021.

[ISZ21] Zachary Izzo, Sandeep Silwal, and Samson Zhou. Dimensionality reduction for wasserstein barycenter. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[IVY19] Piotr Indyk, Ali Vakilian, and Yang Yuan. Learning-based low-rank approximations. In *Advanced in Neural Information Processing Systems (NeurIPS)*, 2019.

[JLL+20] Tanqiu Jiang, Yi Li, Honghao Lin, Yisong Ruan, and David P. Woodruff. Learning-augmented data stream algorithms. In *International Conference on Learning Representations (ICLR)*, 2020.

[KBC+18] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 489–504, 2018.

[KDN+17] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *IJCAI*, 2017.

[KDZ+17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[KLBS+16] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

[LCK18] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[LFKF18] Mario Lucic, Matthew Faulkner, Andreas Krause, and Dan Feldman. Training gaussian mixture models at scale via coresets. *Journal of Machine Learning Research (JMLR)*, 18(160):1–25, 2018.

[LLMV20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

[LMRX21] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In *29th Annual European Symposium on Algorithms (ESA)*, 2021.

[LV18] Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning (ICML)*. PMLR, 2018.

[Mit18] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[Mit20] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, 2020.

[MNS12] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Online optimization with uncertain information. *ACM Transactions on Algorithms (TALG)*, 8(1):1–29, 2012.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2018.

[MSIB21] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, page 105400, 2021.

[MSV+19] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 2019.

[MV20] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.

[NOST18] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[OA92] James B Orlin and Ravindra K Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical programming*, 54(1):41–56, 1992.

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[RBL19] Jack Rae, Sergey Bartunov, and Timothy Lillicrap. Meta-learning neural bloom filters. In *International Conference on Machine Learning (ICML)*, 2019.

[Roh20] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

[SLB+19] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations (ICLR)*, 2019.

[VKKM21] Kapil Vaidya, Eric Knorr, Tim Kraska, and Michael Mitzenmacher. Partitioned learned bloom filter. In *International Conference on Learning Representations (ICLR)*, 2021.

[Wei20] Alexander Wei. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, 2020.

[WLKC16] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[WZ20] Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[XM21] Chenyang Xu and Benjamin Moseley. Learning-augmented algorithms for online steiner tree. *arXiv preprint arXiv:2112.05353*, 2021.

[YTB+21] Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. Path planning using neural a* search. In *International Conference on Machine Learning (ICML)*, 2021.

[ZGA+21] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 258–271, 2021.

A More Related work

Learning-augmented approaches have found success in a wide array of algorithmic tasks. This includes improving classic space complexity in streaming algorithms [HIKV19, IVY19, CGP20, JLL+20, EIN+21, DWM21], and achieving better competitive ratios in online algorithms [MNS12, EKM15, LV18, PSK18, GP19, Roh20, Wei20, LLMV20, BMS20, ACI22, AGP20, WZ20, DKT+21, AGKP21, BCK+22]. Other application domains include data structures [KBC+18, FLV20, Mit18, RBL19, VKKM21], similarity search [WLKC16, DIRW20], and machine scheduling [Mit20, AEMP22, LLMV20, IKMQP21].

A number of works study deep and reinforcement learning for combinatorial optimization and integer programming [BPL+16, KLBS+16, KDN+17, KDZ+17, BDSV18a, NOST18, LCK18, KvHW19, SLB+19, MSV+19, AMW19, AKL+19, ZGA+21, AMK21]. Most of the work in this direction are empirical in nature. See [MSIB21, BLP21] for two recent surveys.

There are also recent works on a related area of data driven algorithm design whose focus is to learn a generalizable algorithm from a family using a small number of samples rather than study how a predictor can aid in an algorithmic task [GR17, BDD+21, BDV18, BDSV18b, BDW18, CGT+20]. See the article [MV20] for a recent survey.

B Omitted Details from Section 3

B.1 Proof of Theorem 3.2

Correctness of Algorithm 1. We now prove the correctness of Algorithm 1. The following three claims are well known facts about the minimum-weight perfect matching problem (see, for instance [AMO93]).

Claim B.1. *There exists a flow of value n if and only if there exists a perfect matching.*

Claim B.2. *If the flow f at the end of the algorithm is a minimum-cost flow of value n, then the returned set of edges is a minimum-cost perfect matching.*

Claim B.3. *A flow f is a minimum-cost flow if and only if G_f contains no negative cost cycles.*

Lemma B.4. *The residual graph G_f at the end of Algorithm 1 contains no cycles which include nodes s or t.*

Proof. At the end of the algorithm, f will have value n and will saturate all edges leaving s as well as all edges entering t. Therefore, all edges containing s will be incoming edges and all edges containing t will be outgoing edges. For both nodes, as all of their edges are oriented in the same direction, they cannot be included in any cycles. □

Lemma B.5. *From their initialization in step 10 of Algorithm 1, all reduced edge costs c' in the residual graph G_f are non-negative.*

Proof. Note that any edges containing s or t always have reduced cost 0. Starting from the costs set at step 10 of the algorithm, we will prove inductively that for all edges (i, j) in the residual graph s.t. $i, j \notin \{s, t\}$, $c'_{ij} \geq 0$. From the feasibility of the dual variables y, for all $(i, j) \in E$ s.t. $i \in L, j \in R$,

$$y_i + y_j \leq c_{ij}.$$

Therefore, for all left to right edges in the residual graph,

$$c'_{ij} = c_{ij} + z_i - z_j = c_{ij} - y_i - y_j \geq 0.$$

It remains to consider edges $(j, i) \in E_f$ s.t. $j \in R, i \in L$. These backwards edges exist if and only if $f_{ij} = 1$. By step 7, $f_{ij} = 1$ only if the matching M derived in step 4 contains (i, j). This implies that $y_i + y_j = c_{ij}$. Therefore, for such edges $(j, i) \in E_f$,

$$c'_{ji} = -c_{ij} + z_j - z_i = -c_{ij} + y_j + y_i = 0.$$

This completes the base case: all edges in the residual graph have non-negative reduced cost at step 10.

For the inductive step, assume going into the while loop that all c'_e for $e = (i,j) \in G_f$ such that $i, j \notin \{s,t\}$ are non-negative. Consider the new reduced costs for any such edge (i,j) at step 13, noting that $d(s,j) \leq d(s,i) + c'_{ij}$. Let c'' denote the updated costs and c' denote the previous reduced costs:

$$\begin{aligned} c''_{ij} = c_{ij} + z_i - z_j &= c'_{ij} + d(s,i) - d(s,j) \\ &\geq c'_{ij} + d(s,i) - (d(s,i) + c'_{ij}) = 0. \end{aligned}$$

So, the new costs are also non-negative.

Now, consider the augmentation in step 17 which will change the edges in the residual graph G_f. In particular, some edges (i,j) will disappear and be replaced by edges (j,i) with reduced costs $c'_{ji} = -c'_{ij}$. As we only augment along edges with reduced costs $c_{ij} = 0$, the reverse edges we create in the residual graph will also have reduced cost 0. Therefore, the invariant holds that costs remain non-negative, completing the proof. □

Lemma B.6. *Assume that there exists a perfect matching. If f has flow value less than n, then the flow g computed in step 16 of Algorithm 1 will have value at least 1.*

Proof. First, note that the flow f maintained by the algorithm will always have capacity in $\{0,1\}$ as the graph G has unit capacities and we always augment by a maximum flow which will have value either 0 or 1 along each edge. If f has value less than n, as the max flow value is n (due to the existence of a perfect matching), there must exist some path of flow value 1 from s to t in G_f. This implies that $d(s,t) > 0$. Let P be the a shortest path from s to t in G_f as measured by the reduced costs c'. Note that for any edge $(i,j) \in P$, $d(s,j) = d(s,i) + c'_{ij}$ (or else, P is not a shortest path). So, after updating the reduced costs in step 13, every edge in P must have reduced cost 0. Therefore, $P \subseteq E'_f$ and the maximum flow computed in step 16 must have flow value 1. □

Lemma B.7. *Algorithm 1 returns a minimum-cost perfect matching, if one exists.*

Proof. Assume that a perfect matching exists. By Lemmas B.1 and B.6, the algorithm will terminate with a flow of value n. By Lemmas B.2 and B.3, it suffices to show that there are no negative cycles in the final residual graph G_f. By Lemma B.4, any cycles in G_f must only contain edges (i,j) where $i, j \notin \{s,t\}$. By Lemma B.5, all reduced costs c'_{ij} of edges $(i,j) \in E_f$ are non-negative. Let $C \subset E_f$ be any cycle in G_f. The value of C is

$$\sum_{(i,j)\in C} c_{ij} = \sum_{(i,j)\in C} c'_{ij} - z_i + z_j \geq \sum_{(i,j)\in C} -z_i + z_j.$$

As C is a cycle, each vertex incident on C has equally many incoming and outgoing edges, cancelling out the contributions of each z_u so that

$$\sum_{(i,j)\in C} -z_i + z_j = 0.$$

Hence, there are no negative cycles in G_f, completing the proof of the correctness of Algorithm 1. □

Runtime of Algorithm 1 We now analyze the runtime of Algorithm 1.

Lemma B.8. *After the first call to maximum cardinality matching in step 4, the partial matching M will contain at least $n - \|y^* - \hat{y}'\|_0$ edges.*

Proof. Consider the minimum-weight perfect matching M^* corresponding to the optimal duals y^*. Note that the edges in M^* are vertex disjoint since they constitute a matching. Let M'' be the set of edges in M^* that are tight under $\hat{y}'$:

$$M'' = \{e = ij \in M^* : \hat{y}'_i + \hat{y}'_j = c_e\}.$$

As y^* is tight for all edges in M^*, if the predicted and optimal duals agree on both endpoints of an edge in M^*, that edge is also tight under $\hat{y}'$. Therefore, out of the n edges in M^*, at most $\|y^* - \hat{y}'\|_0$ of them do not have tight constraints under $\hat{y}'$. Equivalently, $|M''| \geq n - \|y^* - \hat{y}'\|_0$.

Consider the call to maximum cardinality matching in step 4. As M^* is a valid matching and $M'' \subseteq M^*$, M'' is a valid matching as well. In addition, all of M'''s edges are contained in E' (the set of tight edges under $\hat{y}'$). Therefore, M'' is a valid matching for step 4. So, the maximum cardinality matching returned by step 4 must have size at least $|M''| \geq n - \|y^* - \hat{y}'\|_0$, completing the proof. □

Lemma B.9. *The total number of iterations of the while loop in step 11 will be at most* $\|y^* - \hat{y}'\|_0$.

Proof. By Lemma B.6, we will increase the flow value of f by at least 1 each iteration of the while loop. By Lemma B.8, f enters the while loop with value $n - \|y^* - \hat{y}'\|_0$. Therefore, there can be at most $\|y^* - \hat{y}'\|_0$ iterations. □

Lemma B.10. *The total amount of work done by calls to Ford-Fulkerson in step 16 is* $O(m\|y^* - \hat{y}'\|_0)$

Proof. The runtime of Ford-Fulkerson is $O(mf)$ where f is the value of the max flow. By Lemma B.8, the flow value can increase by at most $\|y^* - \hat{y}'\|_0$ over all calls to Ford-Fulkerson before we reach a flow of value n. So, Ford-Fulkerson does a total of $O(m\|y^* - \hat{y}'\|_0)$ work, as required. □

We are now ready to prove the main theorem.

Proof of Theorem 3.2. By Lemma B.7, the algorithm returns a minimum-cost perfect matching, if one exists. It remains to prove the runtime. The first call to matching takes $O(m\sqrt{n})$ time. Constructing the initial flow and residual graph as well as the corresponding costs takes $O(m)$ time. By Lemma B.9, there are at most $\|y^* - \hat{y}'\|_0$ iterations of the while loop. In each iteration, calculating the shortest path distances can be done via Dijkstra's algorithm in $O(m + n \log n)$ time as the reduced costs are always non-negative by Lemma B.5. Updating the costs and constructing the subgraph G'_f takes $O(m)$ time. Augmenting along the flow g takes $O(m)$ time. So, ignoring calls to Ford-Fulkerson, the total running time of work done in the while loop is $O((m + n \log n)\|y^* - \hat{y}'\|_0)$. By Lemma B.10, all calls to Ford-Fulkerson takes a total of $O(m\|y^* - \hat{y}'\|_0)$.

So, the total runtime of the algorithm is $O(m\sqrt{n} + (m + n \log n)\|y^* - \hat{y}'\|_0)$, as required. □

C Improved Learning-Based Minimum-Weight b-Matching

As a corollary to Theorem 3.5, when combined with the near-linear time rounding procedure from [DIL+21], this algorithm gives a fast framework for taking a predicted (possibly infeasible) dual and using it to speed up minimum-weight b-matching.

Corollary C.1. *There exists an algorithm which takes as input a (possibly infeasible) integral dual solution* $\hat{y}$*, produces a feasible dual* $\hat{y}'$ *s.t.* $\|\hat{y}' - y^*\|_{b,1} \leq 5\|\hat{y} - y^*\|_{b,1}$*, and finds a minimum-weight perfect b-matching in* $O(mn + m\|y^* - \hat{y}'\|_{b,0})$ *time, where* y^* *is an optimal dual solution.*

The runtime of $O(mn\|y^* - \hat{y}'\|_{b,1})$ from [DIL+21] is derived from the fact that each time a maximum flow is found in step 13, the flow value is increased by at least 1 (due to the integrality of the problem), and each maximum flow can be found in $O(nm)$ time.

To get the improved runtime in Theorem 3.5, we can follow essentially the same analysis to that for minimum-weight perfect matching, showing that first call to maximum flow will push a significant amount of flow and then bounding the rest of the work in terms of the remaining flow to be pushed.

Proof of Theorem 3.5. The correctness of Algorithm 4 comes from prior work, so it suffices to prove the time complexity. After the first call to max flow in step 6, the flow value will be at least $\sum_i b_i - \|y^* - \hat{y}'\|_{b,0}$ by the same argument as Lemma B.8. In particular, consider an optimal flow (corresponding to a minimum-weight b-matching) f^*. Consider the flow g where $g_e = \min\{f^*_e, u'_e\}$ where u'_e is the capacity of edge e in G' in step 6 of the algorithm (g is the subset of f that satisfies the capacities in G'). For each edge $e = (u, v)$ where

Algorithm 4 Primal-Dual Scheme for MWBM from [DIL+21]

1: **procedure** MWBM-PD($G = (V, E), c, y$)
2: $E' \leftarrow \{ij \in E \mid y_i + y_j = c_{ij}\}$ ▷ Set of tight edges in the dual
3: $G' \leftarrow (L \cup R \cup \{s, t\}, E' \cup \{si \mid i \in L\} \cup \{jt \mid j \in R\})$ ▷ Network of tight edges
4: $\forall e \in E(G')$ s.t. $e = si$ or $e = it$, $u_e \leftarrow b_i$
5: $u_e \leftarrow \infty$ for all other edges of G'
6: $f \leftarrow$ Maximum $s - t$ flow in G' with capacities u
7: **while** Value of f is $< \sum_{i \in L} b_i$ **do**
8: Find a set $S \subseteq L$ such that $\sum_{i \in S} b_i > \sum_{j \in \Gamma(S)} b_j$ ▷ Can be found in $O(m + n)$ time
9: $\epsilon \leftarrow \min_{i \in S, j \in R \setminus \Gamma(S)} \{c_{ij} - y_i - y_j\}$
10: $\forall i \in S$, $y_i \leftarrow y_i + \varepsilon$
11: $\forall j \in \Gamma(S)$, $y_j \leftarrow y_j - \varepsilon$
12: Update E', G', u
13: $f \leftarrow$ Maximum $s - t$ flow in G' with capacities u
14: **end while**
15: $x \leftarrow f$ restricted to edges of G
16: Return x
17: **end procedure**

$y^*_u = y'_u$ and $y^*_v = y'_v$, $g_e = f_e$. Conversely, if there is some vertex u where $y^*_u \neq y'_u$, at worst this vertex can invalidate b_u edges as u can be incident on at most b_u edges in f^*. So, the value of g is at most $\|y^* - \hat{y}'\|_{b,0}$ less than that of f^*. As f^* is optimal, it has value $\sum_i b_i$, and the first call to max flow must push at least $\sum_i b_i - \|y^* - \hat{y}'\|_{b,0}$ units of flow.

Subsequently, over all calls to maximum flow in step 13, the total amount of flow pushed is at most $\|y^* - \hat{y}'\|_{b,0}$ and the total number of iterations of the while loop is at most $\|y^* - \hat{y}'\|_{b,0}$. By implementing max flow in step 13 by the Ford-Fulkerson algorithm, the total amount of work done in the while loop will be $O(m\|y^* - \hat{y}'\|_{b,0})$ as Ford-Fulkerson takes linear time per unit of flow and all other work done in the while loop takes linear time per iteration. As the first call to max flow in step 6 can take time $O(mn)$, we get a total runtime of $O(mn + m\|y^* - \hat{y}'\|_{b,0})$, as required. □

D Omitted Details from Section 4

D.1 Proof of Main Theorem: Theorem 4.1

The goal of the section is to prove the correctness and runtime of Algorithm 2. We first need the following auxiliary lemmas, starting from an observation from [Gol95].

Lemma D.1. *The graph G^- (after contracting all strongly connected components) is acyclic.*

Lemma D.2. *Consider any edge e such that $\ell_{\hat{y}}(e) \geq 0$ at any stage of Algorithm 2. Then e will always continue to satisfy $\ell_{\hat{y}}(e) \geq 0$.*

Proof. Let $e = (u, v)$. We prove the lemma by showing that $\ell_{\hat{y}}(e) \geq 0$ continues to hold after every iteration of the `while` loop in step 3. The only way for $\ell_{\hat{y}}(e)$ to change is if one of y_u or y_v is updated in step 9 of Algorithm 2. If both $u, v \in \cup_{t \geq i^*} L_i$ or both $u, v \notin \cup_{t \geq i^*} L_i$ then the edge is unchanged. Now if $v \in \cup_{t \geq i^*} L_i$ but not u, then $\ell_{\hat{y}}(e)$ increases by 1 so $\ell_{\hat{y}}(e) \geq 0$ continues to hold for this iteration. Now suppose that $u \in \cup_{t \geq i^*} L_i$ but not v. In this case, if $\ell_{\hat{y}}(e)$ was strictly greater than zero, i.e., $\ell_{\hat{y}}(e) \geq 1$, then $\ell_u - \ell_v$ only decreases by 1 so $\ell_{\hat{y}}(e) \geq 0$ is maintained. Lastly we need to consider the possibility that $\ell_{\hat{y}}(e) = 0$. In this case, it must be that $v \in \cup_{t \geq i^*} L_i$ since we can go from x to v via $x \to u \to v$ which means v is either in the same layer as u or possibly a higher layer. Both of these scenarios were addressed previously so we are done. □

Lemma D.3. *At any iteration of the* `while` *loop,* $i^* \leq 2\|\hat{y} - y^*\|_1$ *where the quantity* $\|\hat{y} - y^*\|_1$ *denotes the* ***initial*** *predictor error.*

Proof. We first provide a bound for the *very first* iteration of the `while` loop on Line 3 of Algorithm 2. Note that G^- is acyclic due to Lemma D.1. All paths in G^- must use non-negative edges. Any negative edge $e = (u, v)$ has reduced length at least $-(|y^*_u - \hat{y}_u| + |y^*_v - \hat{y}_v|)$. This is because we know that $\ell(u, v) + y^*_u - y^*_v \geq 0$. Thus, the absolute value of the length of $\ell_{\hat{y}}(e)$ is at most

$$|\ell_{y^*}(e) - \ell_{\hat{y}}(e)| \leq |y^*_u - \hat{y}_u| + |y^*_v - \hat{y}_v|.$$

Now consider the max i for which L_i exists. This means there is a path $x = u_1, \cdots, u_k = v$ of total length $-i$. We have

$$i = \sum_{j=1}^{k-1} |\ell_{\hat{y}}(u_j, u_{j+1})| \leq \sum_{j=1}^{k-1} |y^*_{u_j} - \hat{y}_{u_j}| + |y^*_{u_{j+1}} - \hat{y}_{u_{j+1}}| \leq 2\|y^* - \hat{y}\|_1$$

since every vertex u can appear at most twice in the middle summation above.

We now claim that the maximum i is *always* at most $2\|y^* - \hat{y}\|_1$. Lemma D.2 implies that non-negative edges (under $\ell_{\hat{y}}$) always stay non-negative. Furthermore, the proof of Lemma D.2 tells us that the length of any negative edge is monotonically increasing until the edge becomes non-negative. Therefore any path from x to v in G^- in any iteration of the `while` loop must have also existed in the very first iteration. It follows that most negative distances in G^- are monotonically decreasing every iteration, i.e., becoming less negative. Therefore, the same bound on the number of layers L_i also continues to hold for all instances of G^-. □

Proof of Theorem 4.1. The correctness of Theorem 4.1 follows from the fact that the `while` loop only stops when all reduced edge lengths are non-negative. Therefore, the main challenge is to bound the number of iterations. From the standard analysis of Goldberg's algorithm [Gol95], we know that each iteration of the `while` loop takes $O(m)$ time. This is because G^- is an acyclic graph and thus, finding the layers L_i and all subsequent computations can be done in $O(m)$ time. Thus, it remains to bound the number of `while` loop iterations.

Now call a vertex v touched if $v \in L_{i^*}$ for i^* defined in step 8 of Algorithm 2. Note that for a vertex to be touched, it must exist in some layer and therefore has a negative incoming edge. We now claim that every time a vertex is touched, its most negative *incoming* edge increases in length by $+1$.

Indeed, let (u, v) be the most negative incoming edge to v and suppose that $v \in L_{i*}$. Vertex u cannot exist in layer L_t for some $t > i^*$ since we can consider the path $x \to u \to v$ which implies v must exist in a larger layer than u. Thus when v is touched, the edge length (u, v) must increase by 1. From Lemma D.3, we know that layer L_{i*} has at least $n/(2\|\hat{y} - y^*\|_1)$ many vertices since there are at most $2\|\hat{y} - y^*\|_1$ layers which partition all n vertices. Therefore, at least $n/(2\|\hat{y} - y^*\|_1)$ many vertices get touched in every iteration of the `while` loop. Each vertex can only get touched at most $O(\|y^* - \hat{y}\|_\infty)$ times since the most negative edge length in the very beginning of Algorithm 2 has absolute value at most $O(\|y^* - \hat{y}\|_\infty)$. This implies that the number of `while` loop iterations is at most $O(\|\hat{y} - y^*\|_1 \cdot \|y^* - \hat{y}\|_\infty)$. Since every `while` loop iteration takes $O(m)$ time, the bound of $O(m\|\hat{y} - y^*\|_1 \cdot \|y^* - \hat{y}\|_\infty)$ follows. Note that we could have also used Goldberg's algorithm after getting the reduced edge lengths from $\ell_{\hat{y}}$ with no further modifications to get time $O(m\sqrt{n}\log(\|y^* - \hat{y}\|_\infty)$. Therefore, running these two algorithms in parallel implies the claimed running time. □

D.2 All-Pair Shortest Paths

We observe that Theorem 4.1 implies the following runtime for finding all pairs shortest paths on a graph.

Theorem D.4. *There exists an algorithm which takes as input predicted reduced edge duals* $\hat{y} : V \to \mathbb{Z}$ *and outputs all pair shortest paths in* $O(m \min\{\|\hat{y} - y^*\|_1 \cdot \|\hat{y} - y^*\|_\infty, \sqrt{n}\log(\|\hat{y} - y^*\|_\infty)\} + mn + n^2 \log n)$ *time where* $y^* : V \to \mathbb{Z}$ *denotes a feasible set of reduced edge length duals.*

Proof. Consider Algorithm 2. It applies Algorithm 2 to round $\hat{y}$ into a feasible RE dual $\hat{y}'$. Then we can run Dijkstra's algorithm starting from all vertices in time $O(mn + n^2 \log n)$. The running time follows from Theorem 4.1. □

Algorithm 5 Learning-based Shortest Paths

Input: Graph $G = (V, E)$, predicted duals $\hat{y} : V \to \mathbb{Z}$
procedure FASTER-SHORTEST-PATHS($G, \hat{y}$)
    $\hat{y}' \leftarrow$Round-RE-Duals($G, \hat{y}$) ▷ $\hat{y}'$ is a feasible RE Dual
    for all $v \in V$ **do**
        Run Dijkstra's algorithm starting from v
    end for
    Return all shortest paths found from all vertices
end procedure

E Additional Reductions for Learning-Based Graph Algorithms

E.1 Degree-Constrained Subgraph from Matching

The degree constrained subgraph (DCS) problem is defined as follows. We are given an undirected multigraph $G = (V, E)$ (we will only be considering bipartite graphs) as well as a set of desired upper and lower bound on each vertex's degree: $l_i \leq d_i \leq u_i$ for all $i \in V$. A DCS is an edge-induced subgraph of G where the degree conditions are satisfied. A DCS is called *complete* if each degree achieves its upper bound: $d_i = u_i$ for all $i \in V$.

The maximum perfect DCS and maximum weights DCS problems correspond to maximum perfect matching and maximum weight matching, respectively. Note that the DCS versions of these problems generalize the matching versions by setting $l_i = 0$ and $u_i = 1$ for all vertices. Next, we will show that DCS can also be reduced to matching following the reduction given in [Gab85].

First consider the maximum perfect DCS problem. Let $G = (L, R, E)$ be the corresponding multigraph with degree bounds l_i, u_i for $i \in V$. We will build a corresponding bipartite graph $H = (L', R', E')$ as follows.

- For each vertex i in G, create a complete bipartite graph $K_{\delta,d}$ where $d = d_i$ is the degree of i in G and $\delta = d_i - u_i$ is how many edges need to be removed from i to meet the upper bound. We will call the δ side of $K_{\delta,d}$ *internal* nodes and the d side *external nodes*. Without loss of generality, assume $i \in L$. Then, the external side of $K_{\delta,d}$ is in L' and the internal side is in R'.
- Associate each of i's edges in G with one of its external nodes in H. Specifically, for each $(i, j) \in E$, there will be an edge between one of i's and one of j's external nodes in H and both of those nodes will not be neighbors with any other external nodes. Note that as G is bipartite, with these added edges, H will still be bipartite.
- For each of these external-external edges, give them costs in H corresponding to their costs in G.

First, note that a perfect matching in H corresponds to a perfect DCS in G. For each node i in G, all of its δ internal nodes in H will be matched, meaning that exactly u_i of its external nodes are matched with other external nodes. As each of these external-external edges correspond to edges in the original edgeset E, this means that i will have degree u_i in the subgraph induced by the external-external edges in the perfect matching, as required.

Similarly, it is easy to see that every perfect DCS in G corresponds to a perfect matching in H, so optimizing over perfect matchings/DCS's are equivalent, completing the reduction.

Assume that G had n vertices and m edges (counting copies). In H, we will have $O(m)$ total vertices and $O(m \cdot d_{max})$ total edges where d_{max} is the maximum degree of any vertex in G. Algorithm 3 gives us the following corollary.

Theorem E.1. *Given a maximum weight perfect DCS problem on input graph $G = (V, E)$ with n vertices, m edges, and maximum degree d_{max}, there exists an algorithm which takes takes as input a predicted dual solution $\hat{y}$ to an instance of maximum weight perfect matching derived from G, near-optimally rounds the dual to a feasible solution $\hat{y}'$, and solves the DCS in time $O(m^{3/2}d_{max} + (md_{max} + m \log m)||y^* - \hat{y}'||_0)$.*

E.2 Minimum-Cost 0-1 Flow from Degree-Constrained Subgraph

The reduction bares resemblance to the reduction from shortest path from matching and is also due to [Gab85]. We are given a directed graph G with unit capacities and integral edge costs a_{ij}. We want to find a minimum cost flow of flow value v. We will construct a bipartite multigraph $H = (L, R, E)$ for the DCS problem as follows.

- For each vertex $i \in G$, make two copies $i_1 \in L$ and $i_2 \in R$.
- Add $mindegree(i)$ copies of the edge (i_1, i_2) to H each with weight 0. Where $mindegree(i)$ is the minimum of i's indegree and outdegree.
- For each edge (i, j) in G, add an edge (j_1, k_2) to H with weight $-a_{jk}$.
- Set the degree constrains $u_{i_1} = u_{i_2} = mindegree(i)$ for all $i \neq s, t$. Set $u_{s_2} = mindegree(s)$, $u_{s_1} = u_{s_2} + v$, $u_{t_2} = mindegree(t)$, $u_{t_1} = u_{t_2} + v$.

Note that the number of vertices and edges in H are at most twice those in G.

Theorem E.2. *Given a minimum-cost 0-1 flow problem on input graph $G = (V, E)$ with n vertices, m edges, and maximum degree d_{max}, there exists an algorithm which takes takes as input a predicted dual solution $\hat{y}$ to an instance of maximum weight perfect matching derived from G, near-optimally rounds the dual to a feasible solution $\hat{y}'$, and solves the DCS in time $O(m^{3/2}d_{max} + (md_{max} + m \log m)||y^* - \hat{y}'||_0)$.*

E.3 Diameter to Shortest Paths

The diameter of a graph is defined as the largest distance between any pair of vertices. All exact algorithms for calculating the diameter on general weighted graphs all rely on computing all pairs shortest paths (and there is evidence that this approach is unavoidable [DW21]). Our learning-augmented algorithm for computing shortest-paths of Section 4 gives us the following corollary for computing the diameter of an input graph which follows by first rounding to a valid reduced edge length dual of Definition 4.1 and running all pairs shortest paths using Dijkstra's algorithm on the resulting graph with non-negative weights.

Theorem E.3. *Given an input graph G with n vertices and m edges with possibly negative integer edge lengths given by ℓ, there exists an algorithm which takes takes as input a predicted dual solution $\hat{y}$ to the reduced edge length dual on G and computes the diameter of G in time*

$$O(m \min\{\|\hat{y} - y^*\|_1 \cdot \|\hat{y} - y^*\|_\infty, \sqrt{n} \log(\|\hat{y} - y^*\|_\infty)\}) + \tilde{O}(mn).$$

Remark E.1. *Note that the an algorithm which doesn't use any learned predictions for computing shortest paths on a graph with negative weights, such as the Bellman-Ford algorithm, would have taken time $O(mn^2)$ to compute the diameter. Note that we could have also reduced the diameter problem to matching by using the reduction from shortest paths to matching. However the reduction used in Theorem E.3 is simpler as we don't need to compute any new graphs.*

F Omitted Details from Section 6

Our results generally follow from bounding the pseudo-dimension of the loss function and applying standard uniform convergence for PAC learning.

Definition F.1 (pseudo-dimension)**.** *Let $\mathcal{F}$ be a class of functions $f : X \to \mathbb{R}$. Let $S = \{x_1, x_2, \ldots, x_s\} \subset X$ We say that that S is shattered by $\mathcal{F}$ if there exist real numbers $r_1, \ldots, r_s$ so that for all $S' \subseteq S$, there is a function $f \in \mathcal{F}$ such that $f(x_i) \leq r_i$ if and only if $x_i \in S'$ for all $i \in [s]$. The pseudo-dimension of $\mathcal{F}$ is the largest s such that there exists an $S \subseteq X$ with $|S| = s$ that is shattered by $\mathcal{F}$.*

For a class of loss functions with bounded range and pseudo-dimension, the following lemma provides a PAC learning guarantee.

Lemma F.1 (uniform convergence; e.g., [ABB99])**.** *Let $\mathcal{D}$ be a distribution over a domain X and $\mathcal{F}$ be a class of functions $f : X \to [0, H]$ with pseudo-dimension $d_{\mathcal{F}}$. Consider s i.i.d. samples $x_1, x_2, \ldots, x_s$ from $\mathcal{D}$. There is a universal constant c_0, such that for any $\epsilon > 0$ and $p \in (0, 1)$, if $s \geq c_0 \left(\frac{H}{\epsilon}\right)^2 (d_{\mathcal{F}} + \ln(1/\delta))$, then we have*

$$\left| \frac{1}{s} \sum_{i=1}^{s} f(x_i) - \mathbb{E}_{x \sim \mathcal{D}} f(x) \right| \leq \epsilon$$

for all $f \in \mathcal{F}$ with probability at least $1 - \delta$.

F.1 Proof of Theorem 6.1

To prove Theorem 6.1 Let $f_{1,h}(c) = \ell_1(h, c)$ for $h \in \mathcal{H}$. The following lemma provides a bound on the pseudo-dimension of the family $\mathcal{F}_1 = \{f_{1,h}(c) : h \in \mathcal{H}\}$.

Lemma F.2 ([DIL+21])**.** *The pseudo-dimension of $\mathcal{F}_1$ is bounded above by $O(d \log d)$.*

Now we are ready to prove our learnability result Theorem 6.1.

Proof of Theorem 6.1. Given s samples $c_1, c_2 \cdots, c_s$, the algorithm performs empirical risk minimization on the loss $\hat{\ell}(h) = \sum_{i=1}^{s} \|h^*(c_i) - h\|_1$. The algorithm runs in polynomial time by the efficient optimization assumption.

Moreover, since $\mathcal{H} \subseteq \mathbb{R}^d$ and $\mathcal{H}$ has bounded range, we have that any function in $\mathcal{F}_1$ is bounded by dM. Therefore, the sample complexity and error bound follows from Lemma F.1 and Lemma F.2. □

F.2 Proof of Theorem 6.2

Proof of Theorem 6.2. Similar to the ℓ_1 learnability theorem, the algorithm simply finds the empirical minimizer of $\hat{\ell}(h) = \sum_{i=1}^{s} \|h^*(c_i) - h\|_\infty$. The algorithm runs in polynomial time by the efficient optimization assumption.

Let $f_{\infty,h}(c) = \ell_\infty(h, c)$ for $h \in \mathcal{H}$. It now suffices to bound the pseudo-dimension of the family $\mathcal{F}_\infty = \{f_{\infty,h}(c) : h \in \mathcal{H}\}$ and then apply the uniform convergence lemma (Lemma F.1). Now observe that the pseudo-dimension of $\mathcal{F}_\infty$ can be in turn bounded by the VC dimension of axis-aligned hyperrectangles in $\mathbb{R}^d$, which is known to be $2d$ [MRT18]. □

F.3 Details on Learnability Via Arithmetic Complexity

Suppose we have any loss function $L(h, G) \in \mathbb{R}$ which represents how well a hint vector performs on some input G. For notational simplicity, we define $\mathcal{A}$ as the class of functions in h composed with L:

$$\mathcal{A} := \{L \circ h : h \in \mathcal{H}\}.$$

We also assume that the range of L is equal to $[0, H]$ and that all graphs G can be represented as a feature vector in $\mathbb{R}^m$.

Again, we aim to learn the best function $h \in \mathcal{H}$ which minimizes the following objective:

$$\mathop{\mathbb{E}}_{c \sim D}[L(h, G)]. \tag{3}$$

Towards this end, we let h^* be such the optimal $h \in \mathcal{H}$. We also assume that for each instance G and each $h \in \mathcal{H}$, $L \circ h(G)$ can be computed in time $T(m, d)$. For example, suppose graphs drawn from $\mathcal{D}$ possess edge features in $\mathbb{R}^d$ for some d and our family $\mathcal{H}$ is parameterized by a single vector $\theta \in \mathbb{R}^d$ and represents linear functions which report the dot product of each edge feature with θ. Then it is clear that $T(m, d)$ is a (small) polynomial in the relevant parameters.

The result of this section is to bound the pseudo-dimension of $\mathcal{A}$. After obtaining a bound, we can readily apply Lemma F.1 as we did in the proof of Theorem 6.1 in Section 6.1.

Theorem F.3 (Learnability via computational complexity)**.** *Suppose that any $a \in \mathcal{A}$ takes $T(m, d)$ time to compute given any graph H drawn from $\mathcal{D}$. Then the pseudo-dimension of $\mathcal{A}$ is $O(poly(T(m, d)))$.*

To prove Theorem F.3, we first relate the pseudo-dimension to the VC dimension of a related class of threshold functions. This relationship has been fruitful in obtaining learning bounds in a variety of works such as [LFKF18, ISZ21].

Lemma F.4 (Pseudo-dimension to VC dimension, Lemma 10 in [LFKF18])**.** *For any $a \in \mathcal{A}$, let B_a be the indicator function of the region on or below the graph of a, i.e., $B_a(x, y) = sgn(a(x) - y)$. The pseudo-dimension of $\mathcal{A}$ is equivalent to the VC-dimension of the subgraph class $B_{\mathcal{A}} = \{B_a \mid a \in \mathcal{A}\}$.*

The following theorem then relates the VC dimension of a given function class to its computational complexity, i.e., the complexity of computing a function in the class in terms of the number of operations needed.

Lemma F.5 (Theorem 8.14 in [ABB99])**.** *Let $w : \mathbb{R}^\alpha \times \mathbb{R}^\beta \to \{0, 1\}$, determining the class*

$$\mathcal{W} = \{x \to w(\theta, x) : \theta \in \mathbb{R}^\alpha\}.$$

Suppose that any w can be computed by an algorithm that takes as input the pair $(\theta, x) \in \mathbb{R}^\alpha \times \mathbb{R}^\beta$ and returns $w(\theta, x)$ after no more than r of the following operations:

- *arithmetic operations $+, -, \times$, and $/$ on real numbers,*
- *jumps conditioned on $>, \geq, <, \leq, =$, and $=$ comparisons of real numbers, and*
- *output $0, 1$,*

then the VC dimension of $\mathcal{W}$ is $O(\alpha^2 r^2 + r^2 \alpha \log \alpha)$.

Combining the previous results allows us prove Theorem F.3. At a high level, we are instantiating Lemma F.5 with the complexity of *computing* any function in the function class $\mathcal{A}$.